Eurographics Symposium on Rendering 2010
Jason Lawrence and Marc Stamminger
(Guest Editors)

*Volume 29 (2010), Number 4*

# Adaptive Volumetric Shadow Maps

Marco Salvi†, Kiril Vidimče, Andrew Lauritzen and Aaron Lefohn

Intel Corporation

**Abstract**

*We introduce adaptive volumetric shadow maps (AVSM), a real-time shadow algorithm that supports high-quality shadowing from dynamic volumetric media such as hair and smoke. The key contribution of AVSM is the introduction of a streaming simplification algorithm that generates an accurate volumetric light attenuation function using a small fixed memory footprint. This compression strategy leads to high performance because the visibility data can remain in on-chip memory during simplification and can be efficiently sampled during rendering. We demonstrate that AVSM compression closely approximates the ground-truth correct solution and performs competitively to existing real-time rendering techniques while providing higher quality volumetric shadows.*

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture Coding and Information Theory (H.1.1) [E.4]: Data compaction and compression—Bitmap and framebuffer operations

## 1. Introduction

Realistic lighting of volumetric and participating media like smoke, fog or hair adds significant richness and realism to rendered scenes. Self-shadowing provides important visual cues that define the shape and structure of such media. However, in order to compute self-shadowing in volumetric media, it is necessary to accumulate partial occlusion between visible points and light sources in the scene; doing so requires capturing the effect of all of the volumetric objects between two points and is generally much more expensive than computing shadows from opaque surfaces. As such, while it is common for offline renderers (e.g. those used in film rendering) to compute volumetric shadows, the computation and memory costs required to simulate light transport through participating media have limited their use in real-time applications. Existing solutions for real-time volumetric shadowing exhibit slicing artifacts due to non-adaptive sampling, cover only a limited depth range, or are limited to one type of media (e.g., only hair, only smoke, etc).

This paper introduces a new technique, adaptive volumetric shadow maps (AVSM), that computes approximate volumetric shadows for real-time applications such as games, where predictable performance and a fixed, small memory footprint are required (and where approximate solutions are acceptable). AVSM ignores scattering effects and generates an adaptively-sampled representation of the volumetric transmittance in a shadow-map-like data structure, where each texel stores a compact approximation to the transmittance curve along the corresponding light ray. AVSM can capture and combine transmittance data from arbitrary dynamic occluders, including combining soft media like smoke and very localized and denser media like hair.

† marco.salvi,kiril.vidimce,andrew.t.lauritzen,aaron.lefohn@intel.com

**Figure 1:** *This image shows self-shadowing smoke and hair, both seamlessly rendered into the same adaptive volumetric shadow map. Hair model courtesy of Cem Yuksel.*

The main contribution of our work is the introduction of a streaming lossy compression algorithm that is capable of building a constant-storage, variable-error representation of visibility while the volume is rendered from the light's point of view. We show an implementation of AVSM on current (DirectX 11-class) graphics processors that creates and samples the AVSM at interactive rates. However, this implementation requires a variable amount of memory (proportional to the total number of fragments generated during rasterization) due to limitations of the current DX11 rendering pipeline. We therefore also describe a proof-of-concept software rendering pipeline (running on the GPU) that overcomes these limitations and demonstrates a streaming version of the algorithm with fixed, scene-independent memory requirements.

## 2. Background and Related Work

Adaptive shadowing techniques such as deep shadow maps have been used widely in offline rendering applications [LV00, XTP07]. Deep shadow maps store an adaptive, lossily-compressed representation of the visibility function for each light-space texel. The compression algorithm guarantees a fixed absolute amount of error at the expense of a variable amount of storage and lookup cost.

Many volumetric shadowing techniques used in interactive rendering discretize space into regularly spaced slices that often exhibit banding artifacts [KN01, KPH*03]. Deep opacity maps improve upon opacity shadow maps specifically for hair rendering by warping the sampling positions by the first depth layer [YK08]. Occupancy maps also target hair rendering, use regular sampling but capture many more depth layers than opacity or deep opacity shadow maps by using only one bit per layer, but are limited to volumes composed of occluders with identical opacity [SA09]. Similarly Eisemann et al. compute transmittance of semi-transparent and constant-opacity surfaces via scene voxelization using a one-bit-per-layer slice map [ED06]. Mertens et al. describe a fixed-memory shadow algorithm for hair that adaptively places samples based on a k-means clustering estimate of the transmittance function, assuming density is uniformly distributed within a small number of clusters [MKBvR04]. Recently, Jansen and Bavoil introduced Fourier opacity mapping, which addresses the problem of banding artifacts, but where the detail in shadows is limited by the depth range of volume samples along a ray and may exhibit ringing artifacts [JB10]. FogShop by Zhou et al. [ZHG*07], provides a method for generating single-scattering of low-frequency media such as smoke, and Zhou et al. [ZRL*08] describe a technique for lighting smoke involves a costly pre-computation step making it applicable only to pre-computed animations. Kelley et al. [KGP*94] use a fixed-size staging buffer for sorting fragments. Enderton et al. [ESSL10] introduce a technique for handling all types of transparent occluders in a fixed amount of storage for both shadow and primary visibility, generating a stochastically sampled visibility function.

## 3. Algorithm Overview

Adaptive volumetric shadow maps encode the fraction of visible light from the light source over the interval $[0,1]$ as a function of depth at each texel. This quantity, the *transmittance*, is defined as:

$$t(z) = \mathrm{e}^{-\int_0^z f(x)\,\mathrm{d}x} \tag{1}$$

where $f(x)$ is an attenuation function that represents the amount of light absorbed or scattered along a light ray (see Figure 2).

Given our goal of using a fixed amount of memory, each texel stores a fixed-size array of irregularly-placed samples of the transmittance function. Array elements, the *nodes* of the approximation, are sorted front-to-back, with each node storing a pair of depth and transmittance values $(d_i, t_i)$. Because we adaptively place the nodes in depth, we can represent a rich variety of shadow blockers: from soft and transmissive particles to sharp and opaque occluders. The number of nodes stored per texel is a user-defined quantity, with the only requirement being to store two or more nodes per texel. More nodes allows for a better approximation of transmittance and higher quality shadows, at the expense of increased storage and computational costs.

### 3.1. AVSM Generation

Similarly to standard shadow maps, AVSMs are created by rendering the scene from the light's viewpoint. While standard shadow maps support only infinitely thin opaque occluders, AVSM can handle both opaque and objects of varying thickness and density. Specifically, when a non-opaque occluder is rendered and inserted into the AVSM along a ray from the light, we record the entry and exit points as well as the density along that segment. We analytically integrate the transmittance over the segment and then composite it with

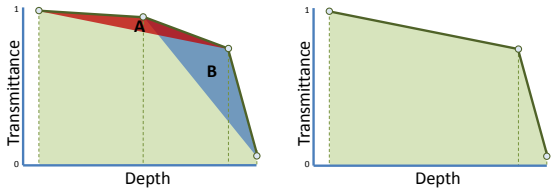


**Figure 2:** *AVSM computes a compressed representation of transmittance along a light ray using an area-based curve simplification scheme. This figure depicts compressing a 4-node curve to 3 nodes. The algorithm removes the node that results in the smallest change in area under the curve, determined by computing the area of the triangle created between the candidate node and its adjacent neighbors (triangles A and B). The right figure shows that we remove the second node from left because triangle A is smaller than triangle B.*

the existing transmittance values. For example, given billboards representing spherical particles, we insert a segment representing the ray's traversal through the particle; for hair we insert a short segment where the light enters and exits the hair strand; and for opaque blockers, we insert a short, dense segment that takes the transmittance to zero at the exit point.

### 3.2. AVSM Compression

In any non-trivial scene the number of light blockers insertions will generate many more nodes than it is possible to store in a shadow map texel (e.g., figure 7 shows a transmittance curve with 282 nodes). When this happens, our algorithm performs an on-the-fly lossy compression of transmittance data, reducing the number of nodes to the maximum node count before proceeding with inserting new blockers or storing the data back to memory. This procedure may require many insertion-compression iterations, hence we need to adopt a lossy compression algorithm that is computationally inexpensive while keeping the overall error small.

Since our compression scheme only needs to remove at most two nodes at a time (which correspond to a single segment insertion), we focus on algorithms that take as input an $n$ node curve and generate an $n-1$ node curve, applying it twice when we need to remove one more node. We note that deep shadow maps-like compression schemes [LV00] operate on the entire uncompressed transmittance curve and alter the transmittance values within a user-defined error threshold, but streaming compression algorithms must not rearrange node positions. This is because over many insertion-compression iterations, nodes can drift unpredictably and perform random walks over the compression plane, leading to non-monotonic transmittance curves and introducing artifacts such as overly dark/bright shadows and temporal aliasing. Given these constraints, AVSM compresses transmittance data simply by removing the node that contributes the least to the overall transmittance curve shape, and the algorithm does not modify node positions.

#### 3.2.1. Transmittance curve approximation

The problem of approximating a polygonal curve $P$ by a simpler polygonal curve $Q$ is of interest in many fields and has been studied in cartography, computer graphics, and elsewhere. Several approximation algorithms have been developed that minimize the approximation error given the number of nodes the approximated curve should have (the so called $min-\varepsilon$ problem) using either distance or area error metrics [DP73, HM88, BCC*06] and with nodes restricted to being a subset of the input nodes [HM88, AV00]. Area-preserving metrics lead to simple and computationally efficient compression code, therefore we compress by removing the node that results in the smallest variation to the integral of the transmittance curve (see Figure 2). We note that area-based metrics are undefined for the first and the last node of the polygonal curve; therefore we apply compression only to internal nodes. In practice, this is beneficial because these uncompressed nodes provide important visual cues such as transition into a volume or the shadows cast from a volume onto opaque surfaces.

Each node of a piecewise transmittance curve maps to an ordered sequence of pairs $(d_i, t_i)$ that encode node position (depth) along a light ray and its associated transmittance. Although transmittance varies exponentially between nodes, like deep shadow maps, we assume linear variation to simplify area computations. This allows us to write the transmittance integral $I_t$ for an $N$ node curve as the sum of $N-1$ trapezoidal areas:

$$I_t = \sum_{i=0}^{N-1} \frac{(d_{i+1} - d_i)(t_i + t_{i+1})}{2}$$

The removal of an internal $ith$ node affects only the area of the two trapezoids that share it. Since the rest of the curve is unaffected we compute the variation of its integral $\triangle_{t_i}$ with a simple geometrically derived formula:

$$\triangle_{t_i} = |(d_{i+1} - d_{i-1})(t_{i+1} - t_i) - (d_{i+1} - d_i)(t_{i+1} - t_{i-1})|$$

### 3.3. AVSM Sampling

Sampling AVSMs can be seen as a generalization of a standard shadow map depth test [Wil78] to soft occluders. Instead of a binary depth test, we evaluate the transmittance function at the receiver depth. This process can be repeated over multiple texels and the results weighted according to a specific reconstruction filter.

Due to the irregular nature of AVSM we cannot rely on texture filtering hardware, so we instead implement filtering manually in the shader. For a given texel we perform a search over the domain of the curve stored in it in order to find the two nodes that bound the shadow receiver of depth $d$, then interpolate the bounding nodes' transmittance $(t_l, t_r)$ to intercept the shadow receiver.

We generally assume the space between two nodes to exhibit uniform density, which implies that transmittance varies exponentially between each depth interval (see equation 1), although we have found linear interpolation to be a faster and visually acceptable alternative:

$$T(d) = t_l + (d - d_l) \cdot \frac{t_r - t_l}{d_r - d_l} \qquad (2)$$

This basic procedure is the basis for point filtering. Bilinear filtering is straightforward: transmittance $T(d)$ is evaluated over four neighboring texels and linearly weighted together.

## 4. Implementation

We implement AVSM and comparative techniques for validation against ground-truth, off-line, and real-time volumetric shadowing techniques. All techniques are implemented

using the DirectX11 rendering and compute APIs. We compare AVSM to real-time techniques, Fourier opacity maps (FOM) and opacity shadow maps (OSM), because these are representative of two important classes of solutions: uniform slicing and frequency-space sampling. For comparison to ground-truth and off-line techniques, we implement a GPU version of deep shadow maps (DSM) as well as an uncompressed version that simply captures and samples from all rendered fragments.

While AVSM is designed to be a streaming compression algorithm, such an implementation requires support for read-modify-write framebuffer operations in the pixel shader. DirectX11 adds the ability to perform unordered read-modify-write operations on certain buffer types in the pixel shader; however, for AVSM we need to ensure that each pixel's framebuffer memory is modified by only one fragment at a time (a per-pixel lock). Because current DX11 compilers forbid per-pixel locks, we implement AVSM in two different ways. First, a variable-memory version that uses the current DX11 rendering pipeline by first capturing all fragments and then compressing. Second, we implement a truly streaming AVSM implementation using a proof-of-concept software particle rasterization pipeline, written in DX11 ComputeShader, that supports the required ordered read-modify-write operations.

AVSM, DSM and the uncompressed solution share a common infrastructure that constructs a linked list of light-attenuating segments per pixel by using DX11's support for atomic gather/scatter memory operations in pixel shaders [YHGT10]. All linked lists were stored in a single buffer, and for our test scenes no more than 20MB of this buffer was typically used. A second pass converts the list of occluding segments at each pixel into a composited transmittance curve—either uncompressed or compressed with the AVSM or DSM compression algorithms.

### 4.1. AVSM Generation and Compression

AVSMs store the transmittance curve as an array of depth-transmittance pairs $(d_i, t_i)$ using two single-precision floating point values. An important ramification of our decision to use a fixed small number of nodes is that the entire compressed transmittance curve fits in on-chip memory during compression. As with classic shadow maps we clear depth to the far plane value, while transmittance is set to 1 in order to represent empty space.

We insert each occluding segment by viewing it as a compositing operation between two transmittance curves respectively representing the incoming blocker and the current transmittance curve. Given two light blockers A and B located along the same light ray, we write the density function $f_{AB}(x)$ as a sum of their density functions $f_A(x)$ and $f_B(x)$. By simply applying equation 1 we can compute their total transmittance:

$$\begin{aligned} t_{tot}(z) &= \mathrm{e}^{-\int_0^z f_{AB}(x)\,\mathrm{d}x} \\ &= \mathrm{e}^{-\int_0^z f_A(x)\,\mathrm{d}x}\mathrm{e}^{-\int_0^z f_B(x)\,\mathrm{d}x} = t_A(z)t_B(z) \end{aligned}$$

In the absence of lossy compression, the order of composition is not important. More relevantly this equation shows that the resulting total transmittance is given by the product of the two transmittance functions respectively associated to each light blocker. Compression proceeds by removing one node at a time until the maximum node count is reached.

In practice, due to the lossy compression, the order in which segments are inserted can affect the results. In particular, in the variable-memory AVSM implementation, the parallel execution of pixel shaders inserts segments into the linked lists in an order that may vary per-frame even if the scene and view are static. Inconsistent ordering can result in visible temporal artifacts, although they are mostly imperceptible and unlikely to be observed when using eight or more nodes or when the volumetric media is moving quickly (e.g., billowing smoke). In those rare cases when a consistent ordering cannot be preserved and the number of nodes is not sufficient to hide these artifacts, we sort the captured segments by depth via insertion sort before inserting them. We discuss the low cost of this sort in Section 5.3.

### 4.2. AVSM Sampling

Determining the light transmittance at a receiver sample requires reconstructing the transmittance curve at its depth. We locate the two nodes that bound the receiver depth via a fast two-level search. Although we must search the irregularly spaced nodes just as with deep shadow maps, the fact that our representation is stored in fixed-sized small arrays results in the memory accesses being coherent and local (no variable-length linked list traversals). In fact, the lookup can be implemented entirely with compile-time (static) array indexing and no dynamic branching, allowing the compiler to keep the entire transmittance curve in registers.

### 4.3. Sofware Particle Rasterization Pipeline

As a proof-of-concept, we demonstrate ordered read-modify-write operations on the frame buffer by building a simple software particle rendering pipeline in DX11 ComputeShader. We start by dividing the screen into tiles and assign each tile to a ComputeShader *threadgroup*. Each threadgroup processes the entire particle set in parallel and builds a list of candidate particles that intersect the tile, ordered by primitive ID. The ComputeShader, now parallelizing over pixels instead of particles, runs the AVSM insertion code for each pixel intersected by a particle. We enforce the correct frame buffer update ordering in this stage by mapping each pixel to a single ComputeShader *thread* (i.e., SIMD lane).


**Figure 3:** *A comparison of smoke added to a scene from a recent game title with AVSM with 12 nodes (left) and deep shadow maps (right). Rendering the complete frame takes approximately 32 ms (30 fps), with AVSM generation and lookups consuming approximately 11 ms of that time. AVSM is 1–2 orders of magnitude faster than a GPU implementation of deep shadow maps and the uncompressed algorithm, yet produce a nearly identical result. Thanks to Valve Corporation for the game scene.*

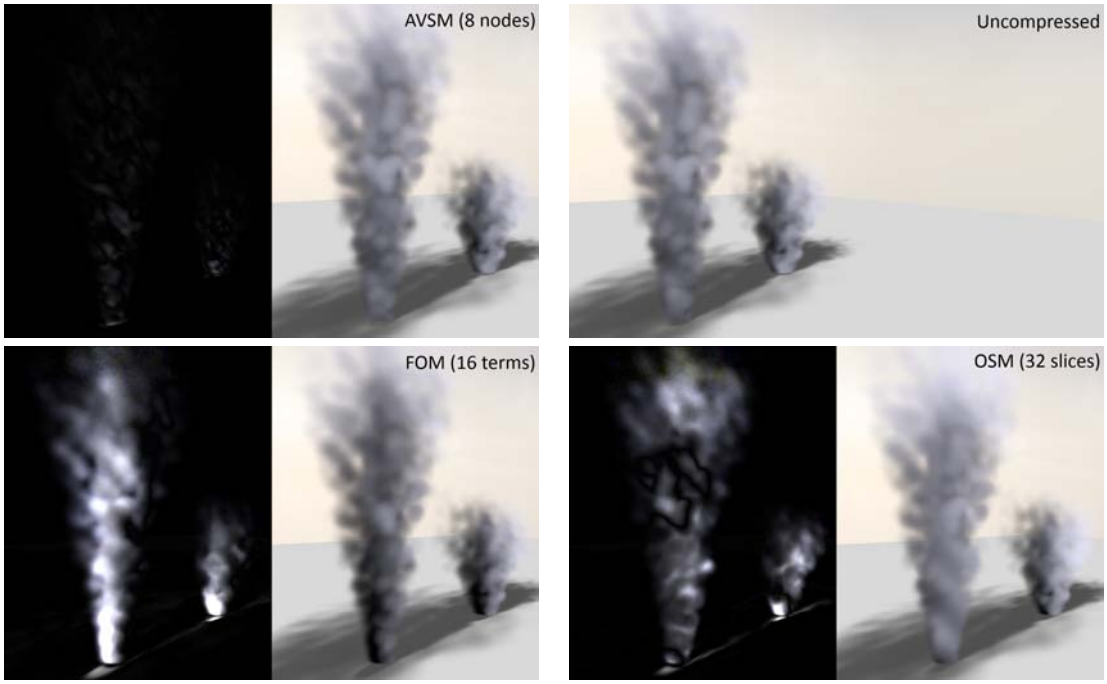


**Figure 4:** *Comparison of AVSM, Fourier opacity maps, and opacity shadow maps to the ground-truth uncompressed result in a scene with three separate smoke columns casting shadows on each other: AVSM with 8 nodes (top left), ground-truth uncompressed (top right), Fourier opacity maps with 16 expansion terms (bottom left), and opacity shadow maps with 32 slices (bottom right). Note how closely AVSM matches the ground-truth image. While the artifacts of the other methods do not appear problematic in these still images, the artifacts are more apparent when animated. Note that the difference images have been enhanced by 4x to make the comparison more clear.*

## 5. Results

All results are gathered on an Intel Core i7 quad-core CPU running at 3.33 GHz running Windows 7 (64-bit) and an ATI Radeon 5870 GPU.

### 5.1. Qualitative Evaluation

Figure 3 shows AVSMs (12 nodes) compared to deep shadow maps (error threshold set to 0.002). There is little perceptible difference between the results, demonstrating that for this real-time scene, our decision to permit variable

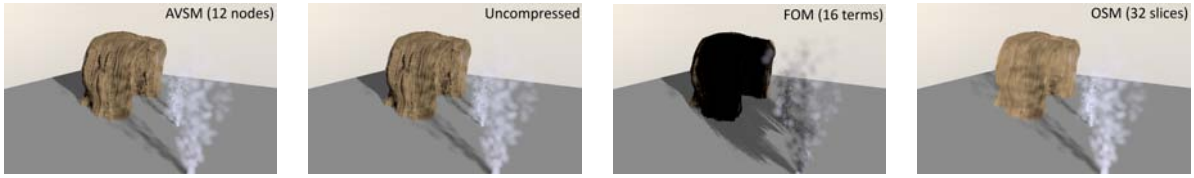


**Figure 5:** *This scene compares (from left to right) AVSM (12 nodes), uncompressed, opacity shadow maps (32 slices), and Fourier opacity maps (16 expansion terms). Note that AVSM-12 and uncompressed are nearly identical and the other methods show substantial artifacts. In particular FOM suffers from severe over-darkening/ringing problems generated by high-frequency light blockers like hair and by less-than-optimal depth bounds. Also note that these images only use bilinear shadow filtering. Using a higher-quality filtering kernel substantially improves the shadow quality. Hair model courtesy of Cem Yuksel.*

error per texel is not a problem. The accuracy of AVSM is further validated by inspecting the transmittance curves and seeing that even with 8 nodes, AVSM very closely approximates the true transmittance curves. The results for sampling the uncompressed data also look identical. Our experience is that 8 nodes results in acceptable visual quality for all views and configurations in this scene. All shadow map sizes in these images are $256^2$.

Figure 4 shows a visual comparison between 8-node AVSM, 16-term Fourier opacity maps, and 32-slice opacity shadow maps all compared against the ground-truth uncompressed result for a scene with three smoke columns casting shadows on each other. Note how much more closely the AVSM matches the ground-truth uncompressed result. The quality improvements are especially noticeable when animated. A key benefit of AVSM compared to these other real-time methods is that AVSM quality is much less affected by the depth range covered by the volumetric occluders.

### 5.2. Quantitative Evaluation

We validate that the AVSM compression algorithm produces accurate results by inspecting a number of transmittance curves and comparing to the ground-truth uncompressed data as well as the deep shadow map compression technique. Overall, we see that the 4-node AVSM shows significant deviations from the correct result, 8-node AVSM matches closely with a few noticeable deviations, and 12-node AVSM often matches almost exactly.

Figure 6 shows a transmittance curve from a combination of smoke and hair (see image in Figure 5) with discrete steps for each blonde hair and smooth transitions in the smokey regions. Note that the 12-node AVSM matches the ground-truth data much more closely than the opacity or Fourier shadow map (both of which use more memory than AVSM to represent shadow data) and is similar to the deep shadow map but uses less memory and is 1–2 orders of magnitude faster. Finally, in Figure 7 very translucent puffs of smoke are encountered close to the light, then there is a sharp drop in transmittance as the second smoke column is encountered, then another drop near the end when the third smoke column is encountered. The 12- and 16-node AVSM produce a close fit to the 282-node uncompressed curve. This type of curve with features spread over a large depth range is difficult or costly to capture with a uniform sampling strategy.

### 5.3. Performance and Memory

AVSM achieves its goal of adaptively sampling volumetric transmittance curves with performance high enough for real-time rendering, achieving up to 85 fps for some smoke configurations, and over 30 fps throughout the Valve scene (Figure 3). Table 1 shows the performance results for the view shown in Figure 4 for AVSM compared to opacity shadow maps, Fourier opacity maps, deep shadow maps the uncompressed approach. For this scene, AVSM compression takes only 0.5–1.5 ms, AVSM lookups take 3–10 ms depending on the number of AVSM nodes, capturing the segments takes 0.4 ms, and sorting the segments before compression takes 3 ms. As discussed earlier, the errors arising from not sorting are often imperceptible and so it can usually be skipped—reducing the AVSM render time to nearly identical that of opacity and Fourier opacity maps.

There are two key sources to AVSM performance. First is the use of a streaming compression algorithm that permits direct construction of a compressed transmittance representation without first building the full uncompressed transmittance curve. The second is the use of a fixed small number of nodes such that the entire representation can fit into on-chip memory. While it may be possible to create a faster deep shadow map implementation than ours, sampling deep shadow maps' variable-length linked lists is costly on today's GPUs and it may result in low SIMD efficiency. In addition, during deep shadow map compression, it is especially challenging to keep the working set entirely in on-chip memory.

Table 1 also shows the memory usage for AVSM, deep shadow maps, and the uncompressed for the smoke scene shown in Figure 4. Note that the memory usage for the variable-memory algorithms shows the amount of memory allocated, not the amount actually used per-frame by the dynamically generated linked lists. The table also shows the

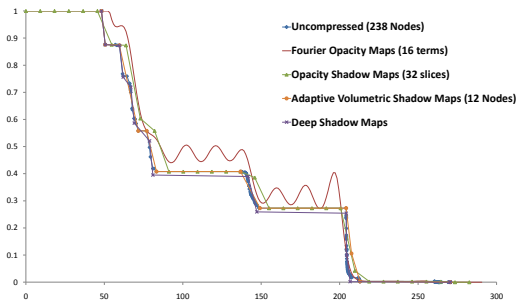


**Figure 6:** *Transmittance curves computed for a scene with a mix of smoke and hair for AVSM (12 nodes) (see Figure 5), Fourier opacity maps (16 expansion terms), opacity shadow maps (32 slices), deep shadow maps, and the ground-truth uncompressed data (238 nodes). The hairs generate sharp reductions in transmittance whereas the smoke generates gradually decreasing transmittance. AVSM matches the ground-truth data much more closely than the other real-time methods.*

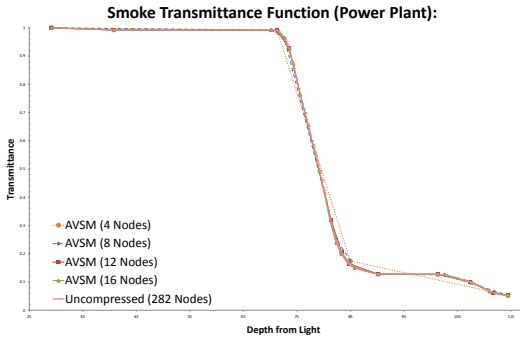


**Figure 7:** *Comparison of the transmittance curves computed by the adaptive volumetric shadow map (AVSM) to the ground-truth uncompressed data set. AVSM with 12 nodes is nearly identical to the uncompressed curve with 282 nodes (top left). The AVSM implementation takes 7–9 ms to generate (including compression). Note the irregularly sampling required to accurately approximate this curve with a small number of samples.*

smaller amount of memory used by the fixed-memory version of AVSM implemented using the modified software particle rendering pipeline.

As described in Section 4, we implemented a proof-of-concept software rendering pipeline running on the GPU that supports pixel shaders that can perform ordered read-modify-write operations on the framebuffer (based on the primitive submission order). While our implementation takes approximately twice as long to generate an AVSM as the fixed pipeline path (including sorting), it nonetheless demonstrates a fixed-memory implementation of AVSM and the benefit of AVSM supporting streaming in-place compression. The memory requirement for this version of AVSM is only 4 MB for a $256^2$ 8-node AVSM.

### 5.4. Limitations

One limitation of AVSM is the introduction of variable error per-texel in exchange for the speed and storage benefits of fixed storage and fast compression. While we show in our test scenes and analysis that this is a valuable trade-off to make for real-time applications insofar that it affords high performance and rarely produces perceptible artifacts, offline rendering users that need absolute quality guarantees may want to continue to use a constant-error compression strategy such as deep shadow maps.

A second limitation is that implementations using current real-time graphics pipelines require a potentially-unbounded amount of memory to first capture all occluding segments along all light rays. In addition, the unordered concurrency in pixel shaders means that when working with a low number of AVSM nodes per texel the segments may need to be re-sorted after capture to eliminate certain temporal artifacts. If future graphics pipelines support read-modify-write memory operations with a stable order, such as ordering by primitive ID, this limitation will go away.

## 6. Conclusions

We have shown that adaptive volumetric shadow maps (AVSM) provide an effective, flexible, and robust volumetric shadowing algorithm for real-time applications. AVSM achieve a high level of performance using a curve simplification compression algorithm that supports directly building the compressed transmittance function on-the-fly while rendering. In addition, AVSM constrains the compressed curves to use a fixed number of nodes, allowing the curves to stay in on-chip memory during compression. As the gap between memory bandwidth and compute capability continues to widen, this characteristic of the algorithm indicates that it will scale well with future architectures.

Although our AVSM implementation using current GPUs requires variable storage, we have prototyped a software graphics pipeline that supports pixel shaders performing ordered read-modify-write operations on the framebuffer while rendering, thereby enabling AVSM to generate compressed transmittance curves while rendering volumetric media from the light in a streaming fashion. In the future, we would like to investigate the performance and hardware implications of providing this ordering option in real-time rendering pipelines to support AVSM and other user-defined streaming compression algorithms.

| | AVSM 4 | AVSM 8 | AVSM 16 | OSM 32 | FOM 16 | DSM 0.01 | Uncompressed |
|---|---|---|---|---|---|---|---|
| Shadow Lookup (Bilinear) | 3 ms | 5.4 ms | 9.5 ms | 1.4 ms | 8.9 ms | 52 ms | 278 ms |
| Capture Occluders | 0.4 ms | 0.4 ms | 0.4 ms | N/A | N/A | 0.4 ms | 0.4 ms |
| Compress | 0.5 ms | 0.7 ms | 1.6 ms | 1 ms | 1.1 ms | 193 ms | N/A |
| Total Frame Time | 9.7 ms | 12.1 ms | 17.43 ms | 8.6 ms | 15 ms | 251 ms | 285 ms |
| Memory Usage | 22(2) MB | 24(4) MB | 28(8) MB | 8 MB | 8 MB | 40 MB | 20 MB |

**Table 1:** *Performance and memory results for* $256^2$ *resolution, adaptive volumetric shadow maps (AVSM) with 4, 8 and 16 nodes, opacity shadow maps (OSM) with 32 slices, Fourier opacity maps (FOM) with 16 expansion terms, deep shadow maps (DSM), and the ground-truth uncompressed data for the scene shown in Figure 4. The AVSM compression algorithm takes 0.5–1.6 ms to build the compressed representation of the transmittance curve even when there are hundreds of occluders per light ray. The optional AVSM sorting cost is 3ms. The total memory required for AVSM and DSM implementations on current graphics hardware is the size of the buffer used to capture the occluding segments plus the size of the compressed shadow map (shown in parentheses). However, our prototype software rendering pipeline generates AVSM using only 2–8 MB.*


## Acknowledgements

We thank Jason Mitchell and Wade Schin from Valve Software for the Left-for-Dead-2 scene and their valuable feedback; and Natasha Tatarchuk and Hao Chen from Bungie and Johan Andersson from DICE for feedback on early versions of the algorithm. Thanks to the the entire Advanced Rendering Technology team, Nico Galoppo and Doug McNabb at Intel for their contributions and support. We also thank others at Intel: Jeffery Williams and Artem Brizitsky for help with art assets; and Craig Kolb, Matt Pharr, Jay Connelly, Elliot Garbus, Pete Baker, and Mike Burrows for supporting the research.


## References


[AV00] AGARWAL P. K., VARADARAJAN K. R.: Efficient algorithms for approximating polygonal chains. *Discrete and Computational Geometry 23*, 2 (2000), 273–291. 3

[BCC*06] BOSEA P., CABELLOB S., CHEONGC O., GUDMUNDSSOND J., VAN KREVELDE M., SPECKMANN B.: Area-preserving approximations of polygonal paths. *Journal of Discrete Algorithms 4*, 4 (2006), 554–566. 3

[DP73] DOUGLAS D. H., PEUCKER T. K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer 10*, 2 (1973), 112–122. 3

[ED06] EISEMANN E., DÉCORET X.: Fast scene voxelization and applications. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2006), ACM, pp. 71–78. 2

[ESSL10] ENDERTON E., SINTORN E., SHIRLEY P., LUEBKE D.: Stochastic transparency. In *I3D '10: Proceedings of the 2010 Symposium on Interactive 3D Graphics and Games* (Feb. 2010), pp. 157–164. 2

[HM88] HIROSHI I., MASAO I.: *Computational-geometric methods for polygonal approximations of a curve*. North-Holland, Amsterdam, 1988. 3

[JB10] JANSEN J., BAVOIL L.: Fourier opacity mapping. In *I3D '10: Proceedings of the 2010 Symposium on Interactive 3D Graphics and Games* (Feb. 2010), pp. 165–172. 2

[KGP*94] KELLEY M., GOULD K., PEASE B., WINNER S., YEN A.: Hardware accelerated rendering of csg and transparency. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM, pp. 177–184. 2

[KN01] KIM T.-Y., NEUMANN U.: Opacity shadow maps. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering* (June 2001), pp. 177–182. 2

[KPH*03] KNISS J., PREMOZE S., HANSEN C., SHIRLEY P. S., MCPHERSON A.: A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (Apr./June 2003), 150–162. 2

[LV00] LOKOVIC T., VEACH E.: Deep shadow maps. In *Proceedings of ACM SIGGRAPH 2000* (July 2000), Computer Graphics Proceedings, ACS, pp. 385–392. 2, 3

[MKBvR04] MERTENS T., KAUTZ J., BEKAERT P., VAN REETH F.: A self-shadowing algorityhm for dynamic hair using clustered densities. In *Rendering Techniques 2004: Eurographics Symposium on Rendering* (Sweden, June 2004), Eurographics / ACM SIGGRAPH Symposium Proceedings, Eurographics. 2

[SA09] SINTORN E., ASSARSON U.: Hair self shadowing and transparency depth ordering using occupancy maps. In *I3D '09: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (Feb./Mar. 2009), pp. 67–74. 2

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 78)* (Aug. 1978), vol. 12, pp. 270–274. 3

[XTP07] XIE F., TABELLION E., PEARCE A.: Soft shadows by ray tracing multilayer transparent shadow maps. In *Rendering Techniques 2007: 18th Eurographics Workshop on Rendering* (June 2007), pp. 265–276. 2

[YHGT10] YANG J., HENSLEY J., GRÜN H., THIBIEROZ N.: Real-time concurrent linked list construction on the gpu. In *Rendering Techniques 2010: Eurographics Symposium on Rendering* (2010), vol. 29, Eurographics. 4

[YK08] YUKSEL C., KEYSER J.: Deep opacity maps. *Computer Graphics Forum 27*, 2 (Apr. 2008), 675–680. 2

[ZHG*07] ZHOU K., HOU Q., GONG M., SNYDER J., GUO B., SHUM H.-Y.: Fogshop: Real-time design and rendering of inhomogeneous, single-scattering media. In *Proceedings of Pacific Graphics 2004* (Nov. 2007), pp. 116–125. 2

[ZRL*08] ZHOU K., REN Z., LIN S., BAO H., GUO B., SHUM H.-Y.: Real-time smoke rendering using compensated ray marching. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers* (New York, NY, USA, 2008), ACM, pp. 1–12. 2